Joint Center for Satellite Data Assimilation

User's Guide to the JCSDA Community Radiative Transfer Model (Beta Version)

October 06, 2005

Yong Han, Paul van Delst , Quanhua Liu,

Fuzhong Weng, Banghua Yan, and John Derber

Joint Center for Satellite Data Assimilation, Camp Springs, Maryland

Table of Contents

List of Tables

List of Figures

1 This Document

This document is a user's guide to the Beta version community radiative transfer model (CRTM), developed at the US Joint Center for Satellite Data Assimilation (JCSDA) with the joint efforts from various academic, private and government research groups. The underline physics and computing algorithms are documented elsewhere (http://www.orbit.nesdis.noaa.gov/smcd/spb/CRTM/index.html) (but a brief overview is given in Section 1.) Here in this document, we describe in details how to use the software.

Document Revision Chart

Version	Primary Author(s)	Description of Version	Date Completed
Beta	Yong Han, Paul van Delst, Quanhua Liu	Created for limited distribution.	October 10, 2005

2 CRTM Overview

2.1 CRTM Components

The development of the CRTM is a community activity involving many scientists from various organizations. For example, four fast radiative transfer solvers have been recently developed: the polarized Delta-4-stream model (Liou et al., 2005), the successive order of integration (SOI) radiative transfer model (Heidinger et al., 2005), the discrete ordinate tangent linear radiative transfer (DOTLRT) (Voronovich et al., 2004), and the advanced doubling-adding (ADA) method (Liu and Weng, 2005). A summary of the development activities are given by Weng et al., (2005). This Beta version should be seen as a starting point of the process in which contributions from various research groups are implemented in the CRTM, evaluated, and then integrated in the JCSDA satellite data assimilation system.

The major components of the Beta version CRTM are shown in Figure 1, followed by brief descriptions of each component.

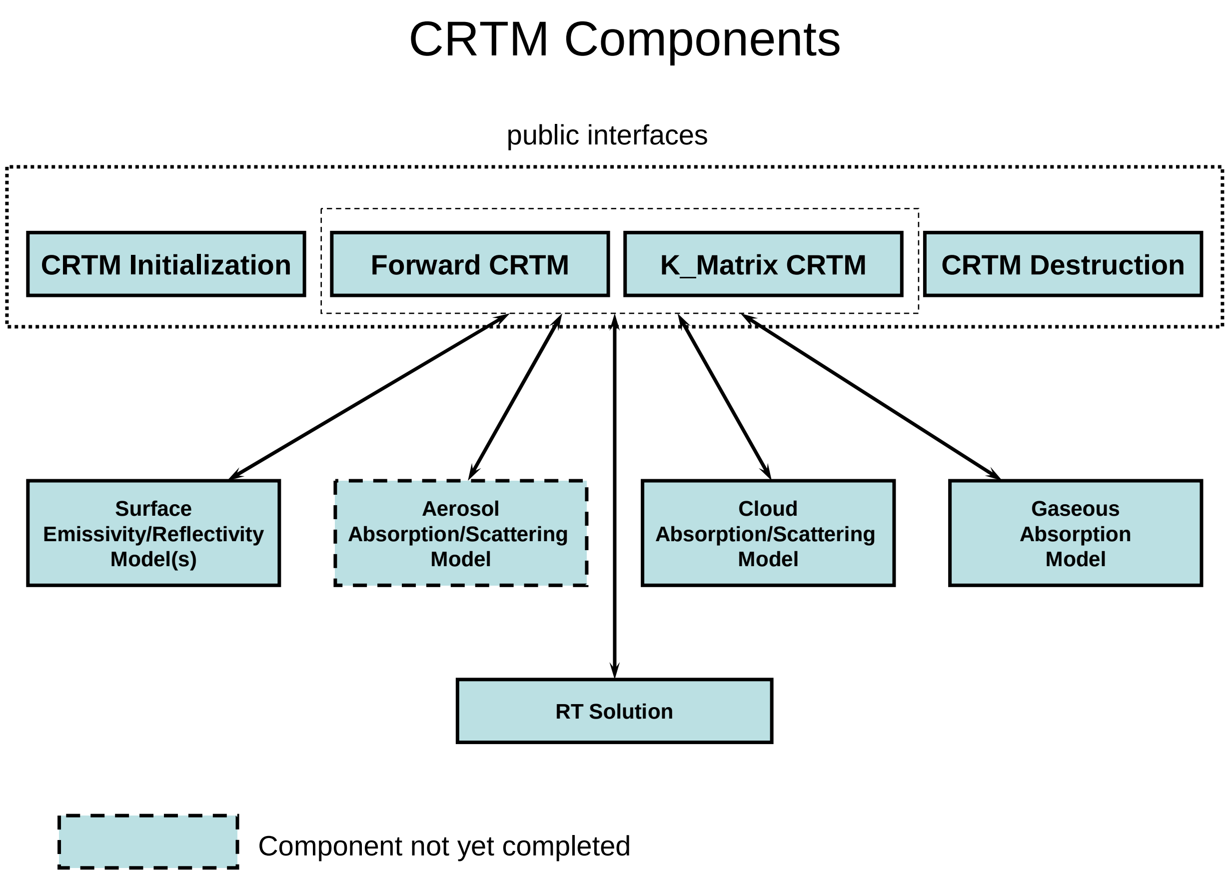

Figure 1. Schematic of the CRTM major components.

Public interface

The CRTM public interface consists primarily of a set of user callable routines (Fortran95 functions and subroutines). For example, the CRTM initialization routine is called to load a set of CRTM data files, which define the set of sensors and channels to be covered in the subsequent calls to the CRTM Forward and K_Matrix (Jacobians) models, and when the model calculations are completed the CRTM destruction routine is called to release the memory occupied by CRTM.

Gaseous absorption model

This component computes gaseous optical depth profile. Currently it is implemented with the compact version of OPTRAN (Optical Path TRANsmittance), due to high efficiency in using computer memory and improved Jacobians. The compact OPTRAN is derived from OPTRAN-v6 and should not be confused with OPTRAN-v7 (see (http://www.orbit.nesdis.noaa.gov/smcd/spb/CRTM/index.html for more detailed description)

OPTRAN currently treats water vapor and ozone as the only variable gases and other absorbing gases as "fixed" gases.

Surface emissivity/reflectivity models

A collection of surface emissivity/reflectivity models are implemented in this version of CRTM. The following is a list of the surface emissivity/reflectivity models.

Microwave:

- Land – LandEM (Weng et al., 2001)
- Snow and sea ice – (Yan & Weng, 2003). For some sensors it offers improved calculations if brightness temperature observations at specified channels are provided.
- Ocean – (Liu and Weng, 2003)

Infrared:

- Ocean – IRSSE (van Delst, 2003; Wu-Smith, 1997)
- Land – measurement database for 24 surface types (NPOESS, Net Heat Flux ATBD, 2001)

In addition, the user may pass his/her own emissivity values to CRTM through the optional arguments of the user interface.

Cloud absorption/Scattering model

This model provides cloud optical parameters such as mass extinction coefficients, single scattering albedo, asymmetric factor and Legendre phase coefficients.

RT solution

The RT solution module solves radiative transfer equation. It is currently implemented with the Advanced Double-Adding (ADA) method (Liu and Weng, 2005).

2.2 Forward, Tangent-linear, Adjoint and K_Matrix Models

CRTM includes the forward and Jacobian models, as well as the Tangent-linear and Adjoint models. The forward model simulates satellite observed radiances. The Jacobian model, often called K_Matrix model, computes radiance derivatives with respect to the input state variables. The K_Matrix model is built through the following pass:

Forward model -> Tangent-linear (TL) model -> Adjoint (AD) model -> K_Matrix (K) model

The Tangent-linear and Adjoint models may be expressed as

$$R\_TL = Hx\_TL \qquad (1)$$
$$x\_AD = H^T R\_AD. \qquad (2)$$

where *x_TL* and *R_TL* are the input and output Tangent-linear variables (vectors) of the state and radiance respectively, while *X_AD* and *R_AD* are the corresponding Adjoint variables. The matrix *H* contains the Jacobian element,

$$\frac{\partial R_i}{\partial x_j},$$

i.e. the derivative of the radiance at the i^{th} channel with respect to the j^{th} state variable, and H^T is the transpose of *H*.

The K_Matrix model may be expressed as

$$X\_K = [h_1 R_1\_K, h_2 R_2\_K, ..., h_m R_m\_K], \quad (3)$$

where R_i_K ($i = 1, m$} is the input *K* radiance variable, a scalar, and h_i is the transpose of the i^{th} row of the *H* matrix:

$$h_i = [\frac{\partial R_i}{\partial x_1}, \frac{\partial R_i}{\partial x_2}, ..., \frac{\partial R_i}{\partial x_n}]^T. \quad (4)$$

Thus, when setting R_i_K = 1 (*i*=1,m), the matrix *X_K* returned from the K_Matrix model contains the Jacobians.

2.3 Software Characteristics

The CRTM is a much more advanced radiative transfer model not only in physics but also in software structure than its predecessor, pCRTM, which is applicable only under clear sky conditions. The software is structurally well modularized - model components are contained in various module program units. Its user interface includes a set of user callable routines (functions and subroutines), allowing the user to integrate CRTM with the user's application program. Derived data types (structure) are used to organize various data and control variables in the user interface. As the CRTM becomes more and more advanced and complex, the uses of structure variables will greatly improve the software clarity and flexibility.

The portability of the software is another important issue and is addressed in part by the uses of the generic kind types defined in model Type_Kinds.f90 model. The floating-point kind type is an especially important one because it is used throughout the CRTM software package in defining the REAL variables.

2.4 Sensor Coverage

Currently the CRTM covers the Microwave and Infrared spectral regions. A list of the sensors and channels for which the CRTM can be applied is presented in Appendix B.

3 CRTM Setup

3.1 Builidng the CRTM library

The first step in building the CRTM library is to unpack the tarball in a CRTM directory,

```
$ cd CRTM
$ tar xvzf JCSDA_CRTM.tar.Z
```

This will create the following directory structure,

```
$ ls –laF
total 964K
-rw-r--r--    1 wd20pd   wd4          920K Oct 21 12:53 JCSDA_CRTM.tar.Z
drwxr-xr-x    2 wd20pd   wd4          4.0K Oct 21 11:49 include/
drwxr-xr-x    2 wd20pd   wd4          4.0K Oct 21 11:50 lib/
-rw-r--r--    1 wd20pd   wd4           21K Oct 21 11:41 make.macros
-rw-r--r--    1 wd20pd   wd4          3.0K Oct 21 11:46 makefile
drwxr-xr-x    3 wd20pd   wd4          4.0K Oct 21 12:15 src/
```

The compilation macros and flags for various systems are contained within the file `make.macros`. The supported platforms and compilers, along with their available makefile targets are,

Platform	Compiler	`makefile` Targets	
		Production (default)	Debug
IBM AIX	`xlf95`	`aix`	`aix_debug`
SunOS	`f95`	`sun`	`sun_debug`**`Error! Bookmark not defined.`**
Linux PGI	`pgf95` (v6.0-5)	`pgi`	`pgi_debug`[1]
Linux Intel	`ifort` (v8.1.023)	`intel`	`intel_debug`
Linux Lahey[2]	`lf95` (v6.2c)	`lahey`	`lahey_debug`
Linux g95	`g95` (Oct 19, 2005)	`g95`	`g95_debug`

Table 1. Supported platforms and compilers for the CRTM build.

The Sun platform has been only minimally tested. The `make.macros` file also contains Linux entries for Absoft and gfortran but these are totally untested.

For IBM and Sun systems, as well as a Linux system with the Lahey compiler, simply typing,

```
$ make
```

[1] The PGI compiler does not successfully build for the debug target. This is being investigated.
[2] This is specified as the default Linux compiler in the `make.macros` file.

should invoke the required compiler. If the build platform is a Linux system and the compiler is one of the others listed in Table 1 , `g95` for example, then typing,

```
$ make g95
```

Continuing with the Linux `g95` example, the output from the make command will look like

```
$ make g95
cd src; make g95; cd ..
make[1]: Entering directory `/usr1/wd20pd/scratch/src'
make -f makefile library "FC=g95" "FL=g95" "FC_FLAGS= -c -O2 -fendian=big -ffree-form -fno-
second-underscore -std=f95  " "FL_FLAGS=   -o" "ENDIAN=Little_Endian"
make[2]: Entering directory `/usr1/wd20pd/scratch/src'
g95  -c -O2 -fendian=big -ffree-form -fno-second-underscore -std=f95   Type_Kinds.f90
g95  -c -O2 -fendian=big -ffree-form -fno-second-underscore -std=f95   File_Utility.f90
g95  -c -O2 -fendian=big -ffree-form -fno-second-underscore -std=f95   Error_Handler.f90
g95  -c -O2 -fendian=big -ffree-form -fno-second-underscore -std=f95   Compare_Float_Numbers.f90
…etc…
```

Note: Currently, invoking the PGI debug target, `make pgi_debug`, will not successfully build the CRTM library. The regular PGI target, `make pgi`, does build successfully. A compiler bug is suspected, but this is still being investigated.

At the end of the compile process, the include files and CRTM library are moved to their respective directories by typing,

```
$ make install
```

The CRTM library can then be used by compiling programs with the switch

```
–ICRTM/include
```

and linking with the switches

```
-LCRTM/lib –lCRTM
```

where the actual location of the CRTM `include` and `lib` directories must be correctly specified for each user's system.

3.2 Compiling the Test Program

A test program tarball, `Test_K_Matrix_software.tar.Z`, has also been provided. To compile this code unpack the tarball and, in the same manner as for the CRTM library build, invoke make,

```
$ tar xvzf Test_K_Matrix_software.tar.Z
$ make
```

Note: The default CRTM include and library directory locations are specified in the test code `makefile` as

```
  INCLUDES=-I$(HOME)/local/CRTM/include
  LIBRARIES=-L$(HOME)/local/CRTM/lib –lCRTM
```

To specify other locations, the file `makefile` must be edited.

The test code tarball also contains instrument coefficient data files for three different sensors,

- NOAA-17 AMSU-A (prefix `amsua_n17`)
- NOAA-17 HIRS/3 (prefix `hirs3_n17`)
- DMSP-16 SSMIS (prefix `ssmis_f16`)

as well as the other data (cloud optical properties, surface emissivity model coefficients, etc) required for the CRTM test.

The included data files are big-endian, unformatted, sequential files and compiler options for the Linux compilers have been included to allow these files to be read on little-endian systems. The one exception is the Lahey compiler which uses run-time options to allow this capability. So, the test program can be invoked by typing

```
$ Test_K_Matrix
```

for any build except that for a Linux platform using the Lahey compiler. In this case, the program is invoked by typing,

```
$ Test_K_Matrix –Wl,-T
```

(note that the last character of `–Wl` is a lower case “ell”, not the number one.)

Each run for a particular instrument will create an output file. For the AMSU-A example this file is called,

```
amsua_n17.CRTM_Test_K-Matrix.output
```

This file can be compared to the baseline output supplied with the test code tarball,

```
amsua_n17.CRTM_Test_K-Matrix.output.Baseline
```

(similarly for the HIRS/3 and SSMIS). Note that small differences in the numbers are to be expected due to differences in the compilers and the options used.

4 CRTM Derived Data Types (Structures)

This section describes the CRTM derived data types or structures (In Fortran90/95, structures are strictly referred to as “derived types,” however the terms structure and derived type will be used interchangeably in this document). Program routines that allocate memory for the structures that have array pointers are also discussed.

4.1 Atmosphere Structure

The Atmosphere structure is used to store data representing the atmospheric state. It is also used as a TL, AD or K variable in the Tangent-linear, Adjoint and K_Matrix model calculations (see Section 2.2).

Structure type name

CRTM_Atmosphere_type

Source module

CRTM_Atmosphere_Define

Example of structure declaration

```
Type(  CRTM_Atmosphere_type ) :: Atmosphere
```

4.1.1 Structure Components

The structure components are described in below in Table 2 and Table 3**.**

Name	Type	Dimension	Initial value	Description
Max_Layers	Integer	Scalar	0	Maximum number of atmospheric layers
N_Layers	Integer	Scalar	0	Number of atmospheric layers.
N_Absorbers	Integer	Scalar	0	Number of atmospheric absorbers (H2O, O3, etc.)
Max_Clouds	Integer	Scalar	0	Maximum number of clouds
N_Clouds	Integer	Scalar	0	Number of clouds
Max_Aerosols	Integer	Scalar	0	Maximum number of aerosol types
N_Aerosols	Integer	Scalar	0	Number of aerosol types
Absorber_ID	Integer pointer	Rank-1 (J)	NULL()	A flag value to identify a molecular species in the absorber profile array (see A1 for valid IDs and discussion #1)

Pressure	Real(fp_kind) Pointer	Rank-1 (L)	NULL()	Layer pressure profile (hPa)
Level_Pressure	Real(fp_kind) Pointer	Rank-1 (0:L)	NULL()	Pressure boundaries of the layer pressure profile (hPa)
Temperature	Real(fp_kind) Pointer	Rank-1 (L)	NULL()	Layer temperature profile (K)
Absorber	Real(fp_kind) Pointer	Rank-2 (L x J)	NULL()	Layer absorber amount profiles (See A1 for valid absorber IDs and units)
Cloud	CRTM_Cloud_type Pointer	Rank-1 (Nc)	NULL()	Structure containing cloud data (see Table 2)
Aerosol	CRTM_Aerosol_type Pointer	Rank-1 (Na)	NULL()	Structure containing aerosol data (see discussion #2), **currently not used**.

Table 2 Description of the Atmosphere structure components. J = number of absorbers, L = number of layers, Nc = number of clouds and Na = number of aerosol types. The initial values are assigned when the structure is declared.

Name	Type	Dimension	Initial value	Description
N_Layers	Integer	Scalar	0	Maximum number of atmospheric layers
Type	Integer	Scalar	NO_CLOUD	Flag value indicating the cloud type (see A2 for the valid cloud types)
Effective_Radius	Real(fp_kind) Pointer	Rank-1 (L)	NULL()	The effective radius of the cloud particle size distribution (microns)
Water_Content	Real(fp_kind) Pointer	Rank-1 (L)	NULL()	The water content of the cloud (kg.m^-2)

Table 3 Description of the Cloud structure components. L = number of layers. The initial values are assigned when the structure is declared.

<u>Discussions</u>

(1) Currently water vapor (H2O) and ozone (O3) are the only valid absorber types. Their symbolic parameter names and units are given in A1. The following is an example of how to set the absorber IDs after the structure variable Atmosphere is declared and allocated:

```
Atmosphere%n_abosrbers = 2
Atmosphere%Absorber_IDs(1) = H2O_ID
Atmosphere%Absorber_IDs(2) = O3_ID
```

Note that, for microwave sensors, the ozone profile in Atmosphere%Absorber(:, 2) does not need to be specified.

(2) Currently the aerosol components in the structure are placeholders. The user should declare the Atmosphere structure as if there are no aerosols (set the argument variable n_Aerosols = 0 when calling the memory allocation routine CRTM_Allocate_Atmosphere, see 4.1.2).

(3) The cloud profile layering scheme is the same as that for temperature and gaseous absorbers.

(4) The Atmosphere structure may contain more than just one cloud, usually clouds of different types.

4.1.2 Memory Allocation Routine

Before populating the Atmosphere structure it must first be allocated using CRTM_Allocate_Atmosphere.

Calling Sequence

```
Error_Status = CRTM_Allocate_Atmosphere( n_Layers,     &
                                         n_Absorbers,  &
                                         n_Clouds,     &
                                         Atmosphere,   &
                                         RCS_Id      = RCS_Id, &
                                         Message_Log = Message_Log )
```

Argument descriptions

The arguments for this function are described in Table 4.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`n_Layers`	`INTEGER`	Scalar or Rank-1 (M)	The number of layers for which there is Atmosphere profile data.
`n_Absorbers`	`INTEGER`	Scalar	The number of gaseous absorbers for which there is Atmosphere profile data.
`n_Clouds`	`INTEGER`	Scalar or Rank-1 (M)	The number of clouds for a particular Atmosphere structure. Can be = 0, i.e. clear sky.
`n_Arosols`	`INTEGER`	Scalar or Rank-1 (M)	The number of aerosols. For now should be set n_Arosole = 0 (the Aerosol component has not yet been completed)
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`Atmosphere`	`CRTM_Atmosphere_type`	Scalar or Rank-1 (M). See Table 4.	Atmosphere structure or structure array with allocated pointer members.
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 4 Description of the CRTM_Allocate_Atmosphere() function arguments and result. M = number of atmospheric profiles.

The dimensionality of the output Atmosphere structure is determined by the dimensionality of the input arguments in the CRTM_Allocate_Atmosphere() function. The allowed input/output argument dimension combinations are shown in Table 5. Note that, currently, the number of absorbers cannot vary with the profile. These multiple interfaces are supplied purely for ease of use depending on what data is available.

Input `n_Layers` **Dimension**	**Input** `n_Absorbers` **Dimension**	**Input** `n_Clouds` **Dimension**	**Output** `Atmosphere` **Dimension**
Scalar	Scalar	Scalar	Scalar
Scalar	Scalar	Scalar	M
M	Scalar	M	M
Scalar	Scalar	M	M
M	Scalar	Scalar	M

Table 5 Allowable dimensionality combinations for the CRTM_Allocate_Atmosphere() function call. M = number of atmospheric profiles.

<u>Content of the structure when it is returned</u>

Atmosphere Component Name	Assigned value	Comments
Max_Layers	n_layers (argument)	These components are assigned with the values from the function's arguments
n_Layers	n_layers (argument)	
n_Absorbers	n_abosrbers (argument)	
Max_Clouds	n_clouds (argument)	
n_Clouds	n_clouds (argument)	
Max_Aerosols	n_aeorsols (argument)	
n_Aerosols	n_aerosols (argument)	
Absorber_ID	INVALID_ABSORBER_ID	
Absorber_Units	INVALID_ABSORBER_UNITS	
Pressure	filled with ZERO	
Level_Pressure	filled with ZERO	
Temperature	filled with ZERO	
Absorber	filled with ZERO	
Cloud	See Table 6	Allocated if n_cloud > 0
Aerosol		Currently not used

Table 6 Values assigned to the Atmosphere structure components by the memory allocation routine CRTM_Allocate_Atmosphere.

The Cloud structure is allocated if the argument n_clouds > 0. When allocated, the structure is assigned with the values listed below in Table 7.

Cloud Component Name	Assigned value	Comments
n_Layers	n_layers (argument)	
Type	NO_CLOUD	
Effective_Radius	Filled with ZERO	
Water_Content	Filled with ZERO	

Table 7 Values assigned to the Cloud structure components by the memory allocation routine CRTM_Allocate_Atmosphere.

4.1.3 Structure Destruction Routine

Calling sequence

```
Error_Status = CRTM_Destroy_Atmosphere( Atmosphere, &
                                        RCS_Id      = RCS_Id, &
                                        Message_Log = Message_Log )
```

Argument Description

The arguments for this function are described below in Table 8.

Name	Type	Dimension	Description
Mandatory Input/Output Arguments			
`Atmosphere`	`CRTM_Atmosphere_type`	Scalar or Rank-1 (M).	Atmosphere structure or structure array.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Optional output arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 8 Description of the `CRTM_Destroy_Atmosphere()` function arguments and result.

4.2 Surface Structure

The Surface structure is used to hold data describing the surface condition in the surface emission and reflection calculations. It is also used as a TL, AD or K variable in the Tangent-linear, Adjoint and K_Matrix model calculations (see Section 2.2).

Structure type name

CRTM_Surface_type

Source module

CRTM_Surface_Define

Example of structure declaration

Type(CRTM_Surface_type) :: Surface

4.2.1 Structure Components

The Surface structure components are described in Table 9 and Table 10. Depending on the surface emission and reflection models, some of the components may not be actually used and therefore can be ignored (see Table 11.)

Name	Type	Dimension	Initial or default value	Description
Gross type of surface determined by coverage				

Land_Coverage	Real(fp_kind)	Scalar	ZERO	Fraction of surface that is of the land surface type
Water_Coverage	Real(fp_kind)	Scalar	ZERO	Fraction of surface that is of the water surface type
Snow_Coverage	Real(fp_kind)	Scalar	ZERO	Fraction of surface that is of the snow surface type
Ice_Coverage	Real(fp_kind)	Scalar	ZERO	Fraction of surface that is of the ice surface type
Land surface type data				
Land_Type	Integer	Scalar	GRASS_SOIL	The land surface type. See A3 for the valid types
Land_Temperature	Real(fp_kind)	Scalar	283.0	The land surface temperature (K).
Soil_Moisture_Content	Real(fp_kind)	Scalar	0.05	The volumetric water content of the soil (g.cm^-3).
Canopy_Water_Content	Real(fp_kind)	Scalar	0.05	The gravimetric water content of the canopy (g.cm^-3).
Vegetation_Fraction	Real(fp_kind)	Scalar	0.3	The vegetation fraction of the surface.
Soil_Temperature	Real(fp_kind)	Scalar	283.0	The soil temperature (K).
Water type data				
Water_Type	Integer	Scalar	SEA_WATER	The water surface type.
Water_Temperature	Real(fp_kind)	Scalar	283.0	The water surface temperature (K).
Wind_Speed	Real(fp_kind)	Scalar	5.0	Surface wind speed (m.s^-1)
Wind_Direction	Real(fp_kind)	Scalar	0.0	Surface wind direction in degree east from North
Salinity	Real(fp_kind)	Scalar	33.0	Water salinity (ppmv)
Snow surface type data				
Snow_Type	Integer	Scalar	NEW_SNOW	The snow surface type. See A3 for the valid types
Snow_Temperature	Real(fp_kind)	Scalar	263.0	The snow surface temperature (K).
Snow_Depth	Real(fp_kind)	Scalar	50.0	The snow depth (mm).
Snow_Density	Real(fp_kind)	Scalar	0.2	The snow density (g.cm^-3)
Snow_Grain_Size	Real(fp_kind)	Scalar	2.0	The snow grain size (mm).
Ice surface type data				
Ice_Type	Integer		FRESH_ICE	The ice surface type.
Ice_Temperature	Real(fp_kind)	Scalar	263.0	The ice surface temperature (K).
Ice_Thickness	Real(fp_kind)	Scalar	10.0	The thickness of the ice (mm)
Ice_Density	Real(fp_kind)	Scalar	0.9	The ice density (g.cm^-3)
Ice_Roughness	Real(fp_kind)	Scalar	ZERO	Measure of the surface roughness of the ice
SensorData containing channel brightness temperatures				
SensorData	CRTM_SensorData_Type	Scalar	See Table 9	Satellite sensor data required for some surface algorithms. Can be left empty.

Table 9. Description of the Surface structure components.

The structure, `SensorData`, a component of the Surface structure, holds satellite sensor measurements for those algorithms that may use the data to improve the results. The SensorData structure components are described below in Table 10.

Name	Type	Dimension	Initial value	Description
n_Channels	Ingeter	Scalar	0	Number of the channels
Sensor_ID	Integer	Scalar	INVALID	WMO sensor ID (see A3.5 for the valid sensor ID)
Tb	Real(fp_kind) pointer	Rank-1 (L)	NULL()	The sensor brightness temperatures (K).

Table 10. Description of the SensorData structure components. L = number of channels.

Discussions

(1) The four surface-coverage variables, Land_Coverage, Water_Coverage, Ice_Coverage and Snow_Coverage must be assigned appropriate fraction values with a sum equal to 1.
(2) When any of the four surface-coverage variables is assigned a non-zero value, the corresponding structure components should be specified, otherwise default values are used. For example, when Water_Coverage > 0, the structure components, Water_Temperature, Wind_Speed and Salinity must be assigned appropriate values if the default values are not the wanted ones in order to correctly use the microwave ocean surface emissivity model.
(3) The set of the structure components under each of the four surface types is a union of the required components used in different surface emissivity/reflectivity. Thus, for a particular surface type and sensor, some of the listed structure components may not be actually used and therefore does not need to be specified. For example, for an ocean surface, the salinity structure component should be specified for a microwave sensor, but may be ignored for an infrared sensor. See Table 11 for descriptions of the requirements for the various surface type components.
(4) Currently only a few microwave sensors are offered with the improved surface emission models that require the brightness temperature measurements, stored in the SensorData structure. See A3.5 for a list of the sensors and conditions. If the SensorData structure component Sensor_ID = INVALID, a default surface emission model will be applied.

Variable Name	Used in the Forward, K_Matrix Model calculations?	
	Microwave Sensor	Infrared Sensor
Land_Coverage	Y	Y
Water_Coverage	Y	Y
Snow_Coverage	Y	Y
Ice_Coverage	Y	Y
Land_Type	N	Y
Land_Temperature	Y	Y
Soil_Moisture_Content	Y	N
Canopy_Water_Content	N	N
Vegetation_Fraction	Y	N
Soil_Temperature	Y	N
Water_Type	N	N
Water_Temperature	Y	Y

Wind_Speed	Y	Y
Wind_Direction	N	N
Salinity	Y	Y
Snow_Type	N	Y
Snow_Temperature	Y	Y
Snow_Depth	Used only for AMSUA/B sensors	N
Snow_Density	N	N
Snow_Grain_Size	N	N
Ice_Type	N	N
Ice_Temperature	Y	Y
Ice_Thickness	Y	N
Ice_Density	N	N
Ice_Roughness	N	N
SensorData	Used for the sensors listed in A3.3	N

Table 11. The surface structure components that are currently used in the surface emissivity models: Y – Actually used, N – not used (no need to be specified)

4.2.2 Memory Allocation Routine

The only component of the Surface structure that needs memory allocation is the `SensorData` structure component – all others are scalars. Thus, if no satellite data is needed for the surface algorithms, or if none is available, then a user can either a) simply not call the Surface allocation function, or b) call the function, but with the `n_Channels` argument set to 0.

<u>Calling Sequence</u>

```
Error_Status = CRTM_Allocate_Surface( n_Channels, &
                                      Surface,    &
                                      RCS_Id      = RCS_Id, &
                                      Message_Log = Message_Log )
```

<u>Argument descriptions</u>

The arguments for this function are described below in Table 12.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`N_Channels`	`INTEGER`	Scalar	The number of channels of satellite data required for some surface algorithms.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`Surface`	`CRTM_Surface_type`	Scalar or Rank-1 (M).	Surface structure or structure array with allocated pointer members.
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 12. Description of the `CRTM_Allocate_Surface()` function arguments and result.

<u>Content of the structure when it is returned</u>

When CRTM_Allocate_Surface is returned, the only changed component of the Surface structure is SensorData structure. It is allocated and assigned the values listed below in Table 13.

SensorData Component Name	Assigned value	Comments
n_Channels	n_layers (argument)	
Sensor_ID	INVALID	
Tb	Filled with ZERO	

Table 13. Values assigned to the SensnorData structure components by the memory allocation routine CRTM_Allocate_Surface().

4.2.3 Structure Destruction Routine

<u>Calling sequence</u>

The Surface structure destruction calling sequence is,

```
Error_Status = CRTM_Destroy_Surface( Surface, &
                                     RCS_Id      = RCS_Id, &
                                     Message_Log = Message_Log )
```

<u>Argument Description</u>

The arguments for this function are described in Table 14.

Name	Type	Dimension	Description
Mandatory Input/Output Arguments			
`Surface`	`CRTM_Surface_type`	Scalar or Rank-1 (M).	Surface structure or structure array.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Optional output arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 14. Description of the CRTM_Destroy_Surface() function arguments and result.

4.3 GeometryInfo Structure

The GeometryInfo structure contains geometry data such as the Earth location, satellite view angle, etc.

Structure type name

CRTM_GeometryInfo_type

Source module

CRTM_ GeometryInfo _Define

Example of structure declaration

Type(CRTM_ GeometryInfo _type) :: GeometryInfo

4.3.1 Structure Components

The structure components currently required are described below in Table 15.

Name	Type	Dimension	Initial value	Description
Sensor_Zenith_Angle	Real(fp_kind)	Scalar	ZERO	The sensor zenith angle (degrees)
Satellite_Height	Real(fp_kind)	Scalar	800km (Default)	Height of the satellite above the Earth surface (for AMSUA/B sensors

				only)
Source_Zenith_Angle	Real(fp_kind)	Scalar	FP_INVALID	Solar zenith angle (for IR sensors)

Table 15. Description of the GeometryInfo structure components.

Discussion

Since the structure components are all scalar, there is no memory allocation routine for the structure.

4.4 ChannelInfo Structure

The ChannelInfo structure holds the sensor channel data, which tell the CRTM model which sensors and channels are included in the subsequent model calculations.

Structure type name

CRTM_ChannelInfo_type

Source module

CRTM_ChannelInfo_Define

Example of structure declaration

Type(CRTM_ChannelInfo_type) :: ChannelInfo

4.4.1 Structure Components

The structure components are described below in Table 16.

Name	Type	Dimension	Initial value	Description
n_Channels	Integer	Scalar	0	Total number of channels.
Channel_Index	Integer pointer	Rank-1 (L)	NULL()	The index of the channels loaded during CRTM initialization (see discussion #2).
Sensor_Channel	Integer pointer	Rank-1 (L)	NULL()	The sensor channel number
Sensor_Descriptor	Character pointer	Rank-1 (L)	NULL()	A character string containing a description of the satellite and sensor sensor name
NCEP_Sensor_ID	Integer pointer	Rank-1 (L)	NULL()	The NCEP/EMC "in-house" value used to distinguish between different sensor/platform combinations.
WMO_Satellite_ID	Integer pointer	Rank-1 (L)	NULL()	The WMO Satellite ID number
WMO_Sensor_ID	Integer pointer	Rank-1 (L)	NULL()	The WMO Sensor ID number

Table 16. Description of the ChannelInfo structure components. L = number of channels

Discussion

(1) Unlike the Atmosphere and Surface structures, whose components are in general require appropriate assignments by the user, the ChannelInfo structure components are filled with the required data by the CRTM initialization routine CRTM_Init or the channel selection routine CRTM_Set_ChannelInfo (see 5.5.1 and 5.5.2).
(2) The Channel_Index array contains the indexes of the channel array loaded during CRTM initialization; the Sensor_Channel array contains the channel numbers for particular sensor(s). In general the two arrays are not the same. For example, Table 17 lists the Channel indexes for the channel array loaded during CRTM initialization. These channels belong to two different sensors, one with 2 channels and the other 4, whose channel numbers are given by the Sensor_Channel array. The Sensor_Descriptor array (or one of the three ID arrays) may be used to identify the sensor to which a particular channel belongs.
(3) The ChannelInfo structure is destroyed when the CRTM_Destroy routine is called (see 5.5.5).

Channel_Index	1	2	3	4	5	6
Sensor_Channel	1	2	1	2	3	4
Sensor_descriptor	Sensor-1	Sensor-1	Sensor-2	Sensor-2	Sensor-2	Sensor-2

Table 17. CRTM channel indexing scheme

4.5 RTSolution Structure

The RTSolution structure is used to hold the results returned from the CRTM calculations. It is also used as a TL, AD or K variable in the Tangent-linear, Adjoint and K_Matrix model calculations (see Section 2.2).

Structure type name

CRTM_RTSolution_type

Source module

CRTM_RTSolution_Define

Example of structure declaration

```
Type(  CRTM_RTSolution_type ) :: RTSolution
```

4.5.1 Structure Components

The RTSolution structure components are listed below in Table 18.

Name	Type	Dimension	Initial value	Description
Radiance	REAL(fp_lind)	Scalar	ZERO	Channel radiance (mW/(m2.sr.cm-1))
Brightness Temperatuer	REAL(fp_lind)	Scalar	ZERO	Brightness temperature (K)
Surface_Emissivity	REAL(fp_lind)	Scalar	ZERO	Surface emissivity at the observation zenith angle
n_Layers	Integer	Scalar	0	Number of layers
Layer_Optical_Depth	Real(fp_kind) pointer	Rank-1 (K)	NULL()	Optional. If this array is allocated, it contains layer total optical depth profile, if not allocated, access this array is an invalid operation.

Table 18. Description of the RTSolution structure. K – number of atmospheric layers

4.5.2 Memory Allocation Routine

The only component of the RTSolution structure that needs memory allocation is the optional `Layer_Optical_Depth` structure component – all others are scalars. It holds the returned optical depth profile. If the optical depth profile is not needed, then a user does not required to call this routine to allocate memory for this component.

<u>Calling Sequence</u>

```
Error_Status = CRTM_Allocate_RTSolution( n_layers, &
                                         RTSolution,    &
                                         RCS_Id      = RCS_Id, &
                                         Message_Log = Message_Log )
```

<u>Argument descriptions</u>

The arguments for this function are described below in Table 19 and Table 20.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`n_Layers`	`INTEGER`	Scalar or Rank-1 (L)	The number of layers for which there is Atmosphere profile data.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`RTSolution`	`CRTM_RTSolution_type`	Scalar or Rank-1 (L or M) or Randk-2 (L x M). See Table 4.	Atmosphere structure or structure array with allocated pointer members.
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 19 Description of the `CRTM_Allocate_RTSolution()` function arguments and result. L – number of channels; M – number of profiles

Input `n_Layers` Dimension	Output `RTSolution` Dimension
Scalar	Scalar, L, M, or L x M
L	L, M, or L x M

Table 20 Allowable dimensionality combinations for the `CRTM_Allocate_RTSolution()` function call. M = number of atmospheric profiles; L – number of channels.

4.5.3 Structure Destruction Routine

<u>Calling sequence</u>

The RTSolution structure destruction calling sequence is,

```
Error_Status = CRTM_Destroy_RTSolution( RTSolution, &
                                        RCS_Id      = RCS_Id, &
                                        Message_Log = Message_Log )
```

<u>Argument Description</u>

The arguments for this function are described in Table 21.

Name	Type	Dimension	Description
Mandatory Input/Output Arguments			
`RTSolution`	`CRTM_RTSolution_type`	Scalar or Rank-1 (L or M) or Rank2 (L x M)	RTSolution structure or structure array.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Optional output arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 21 Description of the CRTM_Destroy_RTSolution() function arguments and result. M = number of atmospheric profiles; L – number of channels.

4.6 Options Structure

The Options structure is used to hold optional input data.

Structure type name

CRTM_Options_type

Source module

CRTM_Options_Define

Example of structure declaration

Type(CRTM_Options_type) :: Options

4.6.1 Structure Components

The structure components are described below in Table 22.

Name	Type	Dimension	Initial value	Description
n_Channels	Integer	Scalar	0	Number of channels.
Emissivity_Switch	Integer	Scalar	0 (NOT_SET)	An integer switch value to indicate emissivity spectrum I/O (See discussion #1)
Emissivity	REAL(fp_kind) pointer	Rank-1 (L)	NULL()	Array to hold the user-supplied

				emissivity spectrum at the observation zenith angle direction (see discussion #2)
Direct_Reflectivity_Switch	Integer	Scalar	0 (NOT_SET)	An integer switch value to indicate direct reflectivity spectrum I/O (See discussion #1)
Direct_Reflectivity	REAL(fp_kind) pointer	Rank-1 (L)	NULL()	Array to hold the user-supplied direct reflectivity spectrum

Table 22 Description of the Options structure. L – number of channels

Discussions

(1) If Emissivity_Switch = 0, the surface emissivity spectrum is computed internally [**DEFAULT**].
If Emissivity_Switch = 1, the surface emissivity spectrum is supplied by the user in the Surface_Emissivity component

If Direct_Reflectivity_Switch = 0, the surface direct reflectivity spectrum used is (1-emissivity) [**DEFAULT**].
If Direct_Reflectivity_Switch = 1, the surface direct reflectivity is spectrum is supplied by the user in the Options%Direct_Reflectivity component

(2) When using the user-supplied surface emissivity, the CRTM assumes a specular surface for MW sensors over ocean and a Lambertian surface for all other situations.

4.6.2 Memory Allocation Routine

Calling Sequence

```
Error_Status = CRTM_Allocate_Options( n_Channels, &
                                      Options,    &
                                      RCS_Id      = RCS_Id, &
                                      Message_Log = Message_Log )
```

Argument descriptions

The arguments for this function are described below in Table 23.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`n_Channels`	`INTEGER`	Scalar	The number of channels used to specify the input spectral data. This value must agree with the number of channels used to define other mandatory spectral CRTM inputs.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`Options`	`CRTM_Options_type`	Scalar or Rank-1	Options structure with allocated pointer members. The rank-1 case is for handling separate Options structures for different profiles. Note that each element of the rank-1 case is allocated to the same number of channels.
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 23 Description of the `CRTM_Allocate_Options()` function arguments and result.

4.6.3 Structure Destruction Routine

Calling sequence

The Options structure destruction calling sequence is,

```
Error_Status = CRTM_Destroy_Options( Options1, &
                                     [Options2, …, Options10], &
                                     RCS_Id      = RCS_Id, &
                                     Message_Log = Message_Log )
```

Argument Description

The arguments for this function are described in Table 24.

Name	Type	Dimension	Description
Mandatory Input/Output Arguments			
`Options1, [Options2, …, Options10]`	`CRTM_Options_type`	Scalar or Rank-1	Structure(s) to be re-initialized. At least one structure or structure array must be specified and no more than 10 structures or structure arrays must be specified.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Optional output arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 24 Description of the CRTM_Destroy_Options() function arguments and result.

5 CRTM User Interface

5.1 CRTM Floating-Point Kind Type

Fortran 90/95 allows users a new methodology to specify the precision of intrinsic data types, that is, specifying what “kind” of intrinsic type you want. Although we have defined several kind types for integer and floating-point (real) data types in the Type_Kinds.f90 module, the following discussion is limited to the floating-point kind type, fp_kind, since it has been applied throughout the CRTM code where floating point variables and parameters are defined[3].

Consider the following example of a kind definition module for real numbers only, and a simple program that uses it,

```
MODULE type_kinds
  IMPLICIT NONE
  INTEGER, PARAMETER :: Single = SELECTED_REAL_KIND( 6 )
  INTEGER, PARAMETER :: Double = SELECTED_REAL_KIND( 15 )
  INTEGER, PARAMETER :: fp_kind = Double
END MODULE type_kinds

PROGRAM test_kinds
  USE type_kinds
  REAL( fp_kind ) :: x, y
  x = 1.0_fp_kind
  y = x + 1.0_fp_kind
END PROGRAM test_kinds
```

By using the generic kind type in the program, rather than the specific types, `Single` or `Double`, the precision of numbers used in the program can be changed simply by changing the definition in the `type_kinds` module, and recompiling. This can be useful in identifying sections of an algorithm that are sensitive to the numerical precision of the variables it uses and eliminate the mixing of precisions of variables (e.g. in expressions containing both single and double precision floating point numbers) since this can introduce numerical instabilities into algorithms. More importantly it ensures the program being portable to various computer platforms, which may have different hardware presentations for numerical computation. We strongly recommend the user to the kind type fp_kind to declare all the floating-point variables that are interacted with the CRTM user interface.

5.2 Atmosphere Profile Layering Scheme

CRTM requires layer atmospheric profiles, that is, variables such as temperature and water vapor do not vary in the layer. The vertical coordinates is given by the layer pressures, accompanied with the level pressures describing the layer boundaries. This layering scheme is illustrated in

[3] An exception to this rule is for data file I/O where specific kind types are used.

Figure 2. The profile data are stored in arrays with the pressure data in ascending order. The first element of the level pressure array is indexed with an integer value 0, but for the arrays holding the layer quantities the index value starts at 1. The CRTM does not require a fixed number of layers and layer thicknesses, but it does require that the number of layers do not exceed MAX_N_LAYERS (currently 100), defined in module CRTM_Parameters.f90 and the user need to set the top pressure level at 0.005 hPa to be consistent to the coefficient data set of the gaseous absorption model. It is the user's responsibility to supply a meaningful atmospheric profile.

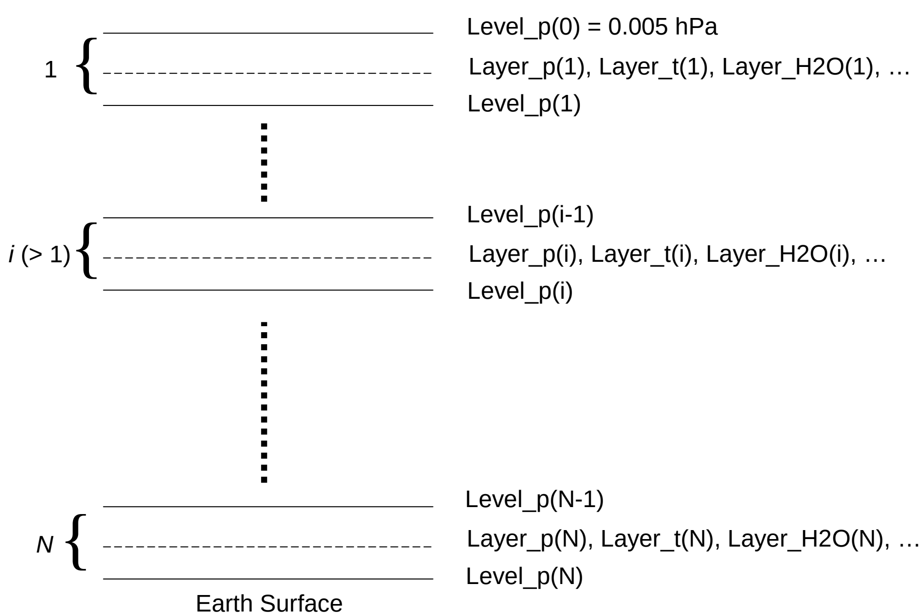

Figure 2 Atmosphere profile layering scheme

5.3 CRTM Coefficient Data files

The CRTM computing algorithms rely heavily on a set of pre-prepared data sets. These data are stored in several binary files, and are all loaded into the CRTM data variables during the CRTM initialization phase. These files may be divided into two groups: one with data specific to a collection of sensors and the other valid for all sensors or the sensors in a whole spectral region such as Microwave and Infrared. The sensor-sensitive data files determine the range of the sensors for which the CRTM can be applied. For example, if the files contain data for one sensor only, calling CRTM model for other sensors would be an illegal operation. The sensor-sensitive files are originally generated with one sensor per file, but the user can concatenate them into files that work for multiple sensors by using the tools included in the CRTM software package. In the following we provide only brief introductions of the data files, since knowing their details is not necessary.

Spectral coefficient (SpcCoeff) file

The file contains spectral coefficient data about particular sensors, including satellite and sensor identifications, sensor type (microwave, infrared, or visible), channel numbers and channel central frequencies. A collection of the SpcCoeff files, each containing data for one sensor only, has been archived. The user may concatenate them into one file for the desired sensors using the CRTM utility tools. The CRTM loads only one SpcCoeff file. The coefficient data files and the software to concatenate them can be obtained on line at http://cimss.ssec.wisc.edu/~paulv/Fortran90/CRTM/Developmental

Optical depth (TauCoeff) coefficient file

The file contains coefficient data specific to a set of sensors for use in gaseous optical depth calculations. It must be consistent with the SpcCoeff file in the range of the sensors included. A collection of the SpcCoeff files, each containing data for one sensor only, has been archived. The user may concatenate the files for a set of desired sensors into a single file using the CRTM utility tools. The CRTM loads only one TauCoeff file. The coefficient data files and the software to concatenate them can be obtained on line at http://cimss.ssec.wisc.edu/~paulv/Fortran90/CRTM/Developmental

Cloud coefficient (CloudCoeff) file

The file contains cloud optical parameters and lookup tables such as mass extinction coefficients, single scattering albedo, asymmetry factors and Legendre expansion coefficients. These data are not sensor specific.

Surface Emissivity coefficient (EmisCoeff) file

The file currently contains coefficient data for computing infrared ocean surface emissivity. These data are not sensor specific.

Aerosol coefficient (AerosolCoeff) file

Currently it is a dummy file, used as a placeholder.

5.4 Error codes and message handling

In an effort to introduce some form of graceful error traceback utility, the `error_handler.f90` module is provided. This module defines simple error codes and contains a routine to output messages. The available error code parameters are shown in Table 25.

Error Code Parameter	Description
`SUCCESS`	Specifies successful completion.
`INFORMATION`	Specifies information output.
`WARNING`	Specifies a warning state. Execution can continue but results may be incorrect.
`FAILURE`	Specifies a sever, unrecoverable error. Execution cannot continue.
`UNDEFINED`	Specifies an undefined status.

Table 25. Error code parameters provided in the Error_Handler module.

The `Error_Handler` module also contains the `Display_Message` subroutine. This routine allows message to be displayed on standard output (screen, the default), or written to a user specified log file. An example of its usage is shown below,

```
USE Error_Handler
CHARACTER( * ), PARAMETER :: ROUTINE_NAME = 'My_Routine'
INTEGER :: Error_Status

Error_Status = calculate_widget_size()
IF ( Error_Status /= SUCCESS ) THEN
  CALL Display_Message( ROUTINE_NAME, &
                        'Error calculating widget size., &
                        Error_Status, &
                        Message_Log = 'error_log.txt' )
  RETURN
END IF
```

The output message format is,

"routine name"("state description") : "message"

For example, if an error occurs in the `Display_Message` routine attempting to write to the log file, the output is,

```
DISPLAY_MESSAGE(FAILURE) : Error opening message log file
```

5.5 User Interface Routines

We have introduced the memory allocation and deallocation routines in Section 4. The user is strongly recommended to use these routines to allocate/deallocate memory for the structures that have pointer components. In this subsection we describe the rest of the user interface routines.

5.5.1 CRTM Initialization Routine CRTM_Init

The CRTM initialization routine is used to initialize the CRTM before the CRTM Forward or K_Matrix models are called. It loads the required coefficient data and sets the initial content of the ChannelInfo structure.

<u>Calling sequence</u>

```
Error_Status = CRTM_Init( ChannelInfo,                        &
                               SpcCoeff_File = SpcCoeff_File,        &
                               TauCoeff_File = TauCoeff_File,        &
                           AerosolCoeff_File = AerosolCoeff_File,    &
                             CloudCoeff_File = CloudCoeff_File,      &
                              EmisCoeff_File = EmisCoeff_File,       &
                           Sensor_Descriptor = Sensor_Descriptor,    &
                              NCEP_Sensor_ID = NCEP_Sensor_ID,       &
                              Sensor_Channel = Sensor_Channel,       &
                                   File_Path = File_Path,            &
                                       Quiet = Quiet,                &
                                  Process_ID = Process_ID,           &
                           Output_Process_ID = Output_Process_ID,    &
                                      RCS_Id = RCS_Id,               &
                                 Message_Log = Message_Log        )
```

Argument descriptions

The arguments for this function are described below in Table 26.

Name	Type	Dimension	Description
		Optional Input Arguments	
SpcCoeff_File	CHARACTER(*)	Scalar	Name of the CRTM Binary format SpcCoeff file. If not specified, “SpcCoeff.bin” is the default.
TauCoeff_File	CHARACTER(*)	Scalar	Name of the CRTM Binary format TauCoeff file. If not specified, “TauCoeff.bin” is the default.
AerosolCoeff_File	CHARACTER(*)	Scalar	Name of the CRTM Binary format ScatterCoeff file. If not specified, “ScatterCoeff.bin” is the default.
CloudCoeff_File	CHARACTER(*)	Scalar	Name of the Binary format CloudCoeff file. If not specified, “CloudCoeff.bin” is the default.
EmisCoeff_File	CHARACTER(*)	Scalar	Name of the Binary format EmisCoeff file. If not specified, “EmisCoeff.bin” is the default.
File_Path	CHARACTER(*)	Scalar	Path for the input coefficient files. If not specified, the current directory is the default.
Sensor_Descriptor	CHARACTER(*)	Rank-1	List of satellite/sensor descriptors for each channel the user wants to process. If not specified, all the channels defined by the SpcCoeff and TauCoeff data will be selected in ChannelInfo. (see Appendix B for a list of the sensor descriptors)
NCEP_Sensor_Id	INTEGER	Rank-1	List of NCEP sensor ids for each channel the user wants to process. If not specified, all the channels defined by the SpcCoeff and TauCoeff data will be selected in ChannelInfo. Ignored if the `Sensor_Descriptor` argument is passed.
Sensor_Channel	INTEGER	Rank-1	List of channel numbers for each sensor the user wants to process. Used with

			either the `Sensor_Descriptor` or `NCEP_Sensor_Id` argument, and ignored if neither of these are present.
`Quiet`	`INTEGER`	Scalar	Set (1) this argument to *suppress* `INFORMATION` message output. By default (0), `INFORMATION` messages are output. If `Quiet` = 0, messages are output If `Quiet` = 1, messages are suppressed.
`Process_Id`	`INTEGER`	Scalar	Set this argument to the MPI process ID that this function call is running under. If MPI is not being used, ignore this argument. This argument is ignored if the `Quiet` argument is set.
`Output_Process_Id`	`INTEGER`	Scalar	Set this argument to the MPI process ID in which all `INFORMATION` message are to be output. If this argument is used, the `Process_Id` argument must also be used. This argument is ignored if the `Quiet` argument is set.
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`ChannelInfo`	`CRTM_ChannelInfo_type`	Scalar	Structure containing indexing information for the sensor(s) and channel(s) the user requested.
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 26. Description of the `CRTM_Init` function arguments and result.

<u>Discussion</u>

(1) The ChannelInfo structure is set according to the user supplied information:
- If the optional argument pairs (Sensor_Descriptor, Sensor_Channel) and (NCEP_Sensor_Id, Sensor_Channel) are not specified, all the channels registered in the the SpcCoeff and TauCoeff data files will be selected;
- If (Sensor_Descriptor, Sensor_Channel) or (NCEP_Sensor_Id, Sensor_Channel) is specified, the channel selection is based on the channel numbers given by Sensor_Channel and the sensor/satellite platform IDs given by Sensor_Descriptor or NCEP_Sensor_Id.

(2) The user does not need to allocate the ChannelInfo structure. CRTM_Init will allocate the structure no matter it is already allocated or not.

5.5.2 Channel Selection Routine CRTM_Set_ChannelInfo

After the CRTM initialization, the user may need to change the set of sensors and channels. The function CRTM_Set_ChannelInfo may be called to reset the content of the structure ChannelInfo.

<u>Calling sequence</u>

Select a sensor with all its available channels:

```
Error_Status = CRTM_Set_ChannelInfo( Sensor_Descriptor, &
                                     ChannelInfo,       &
                                     RCS_Id = RCS_Id,   &
                                     Message_Log = Message_Log )
```

or select a set of available channels:

```
Error_Status = CRTM_Set_ChannelInfo( Sensor_Descriptors,  &
                                     Sensor_Channels,     &
                                     ChannelInfo,         &
                                     RCS_Id = RCS_Id,     &
                                     Message_Log = Message_Log )
```

<u>Argument descriptions</u>

The arguments for this function are described below in Table 27.

Name	Type	Dimension	Description
Input Arguments			
`Sensor_Descriptor`	`CHARACTER(*)`	Scalar	A satellite/sensor descriptor (see Appendix B for a list of the sensor descriptors and the following discussion)
`Sensor_Descriptors`	`CHARACTER(*)`	Rank-1 (channel dimension)	A list of satellite/sensor descriptors, used to explain each of the channels listed in Sensor_Channels. (see Appendix B for a list of the sensor descriptors and the following discussion)
`Sensor_Channels`	`INTEGER`	Rank-1 (channel dimension)	List of channel numbers, used with Sensor_Descriptors (see discussion)
`Process_Id`	`INTEGER`	Scalar	Set this argument to the MPI process ID that this function call is running under. If MPI is not being used, ignore this argument. This argument is ignored if the `Quiet` argument is set.
Optional Input Arguments			
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Output Arguments			
`ChannelInfo`	`CRTM_ChannelInfo_type`	Scalar	Structure containing indexing information for the sensor(s) and channel(s) the user requested.
Optional Output Arguments			

`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 27 Description of the `CRTM_Set_ChannelInfo()` function arguments and result.

<u>Discussion</u>

Example 1: to request HIRS3 on NOAA-16 and all its channels, one may call the function as

Error_status = CRTM_Set_ChannelInfo('hirs3_n16", ChannelInfo)

where "hirs3_n16" is the sensor's descriptor.

Example 2: if only channels 8, 9, 10 are needed, one may specify ChannelInfo as

Sensor_descriptors(:) = "hirs3_n16"
Sensor_Channels (:) = (/8, 9, 10/)
Error_status = CRTM_Set_ChannelInfo(Sensor_descriptors, sensor_Channels, ChannelInfo)

where Sensor_descriptors and Sensor_Channels are both three-element arrays.

5.5.3 Forward Model Routine CRTM_Forward

The CRTM forward model is provided to simulate satellite radiometric observations.

<u>Calling sequence</u>

```
Error_Status = CRTM_Forward( Atmosphere,   &
                             Surface,      &
                             GeometryInfo, &
                             ChannelInfo,  &
                             RTSolution,   &
                             Options      = Options, &
                             RCS_Id       = RCS_Id,  &
                             Message_Log = Message_Log )
```

<u>Argument description</u>

The arguments for this function are described below in Table 28.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`Atmosphere`	`CRTM_Atmosphere_type`	Scalar or Rank-1 (M)	Atmopsheric state profile data. If rank-1, M = number of profiles.
`Surface`	`CRTM_Surface_type`	Scalar or Rank-1 (M)	Surface state data. If rank-1, M = number of profiles.
`GeometryInfo`	`CRTM_GeometryInfo_type`	Scalar or Rank-1 (M)	View geometry data. If rank-1, M= number of profiles.
`ChannelInfo`	`CRTM_ChannelInfo_type`	Scalar	ChannelInfo structure returned in the initialization function or the channel selection function call.
Optional Input Arguments			
`Options`	`CRTM_Options_type`	Scalar or Rank-1 (M)	Options structure containing the option arguments for the CRTM. The dimension should be the same as the input Atmosphere structure.
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`RTSolution`	`CRTM_RTSolution_type`	Rank-1 (L) or Rank-2 (L×M)	Solution to the RT problem for each requested channel (L), and each input profile (M). If the input data is scalar, this output is rank-1 (L). If the input data is rank-1 (M), this output is rank-2 (L×M).
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 28 Description of the `CRTM_Forward()` function arguments and result. L = number of channels; M = number of profiles.

<u>Discussions</u>

(1) The contents of the input structures Atmosphere, Surface, GeometryInfo and ChannelInfo must be appropriately specified. See Section 4 for their definitions and valid assignments.
(2) The structure component RTSolution%Layer_Optical_Depth is an array pointer and thus must be allocated if the user requests the CRTM to return the total optical depth profile; it is ignored if the user does not allocate the memory.

5.5.4 Jacobian Routine CRTM_K_Matrix

The Jacobian routine CRTM_K_Matrix computes radiance or brightness temperature derivatives with respect to the input state variables.

Calling sequence

```
  Error_Status = CRTM_K_Matrix( Atmosphere,     &
                                     Surface,     &
                                RTSolution_K,     &
                                GeometryInfo,     &
                                 ChannelInfo,     &
                                Atmosphere_K,     &
                                   Surface_K,     &
                                 RTSolution,      &
                               Options      = Options, &
                                     RCS_Id = RCS_Id,  &
                                Message_Log = Message_Log )
```

Argument description

The arguments for this function are described below in Table 29.

Name	Type	Dimension	Description
Mandatory Input Arguments			
`Atmosphere`	`CRTM_Atmosphere_type`	Scalar or Rank-1 (M)	Atmopsheric state profile data. If rank-1, M = number of profiles.
`Surface`	`CRTM_Surface_type`	Scalar or Rank-1 (M)	Surface state data. If rank-1, M = number of profiles.
`RTSolution_K`	`CRTM_RTSolution_type`	Rank-1 (L) or Rank-2 (L x M)	Structure containing the RT solution K-matrix inputs. NOTE: (1) RTSolution_K%Surface_Emissivity and RTSolution_K%Layer_Optical_Depth are not defined and therefore should be ignored; (2)on EXIT from this function, the contents of this structure may be modified (e.g. set to zero.)
`GeometryInfo`	`CRTM_GeometryInfo_type`	Scalar or Rank-1 (M)	View geometry data. If rank-1, M= number of profiles.
`ChannelInfo`	`CRTM_ChannelInfo_type`	Scalar	ChannelInfo structure returned in the initialization function call.
Optional Input Arguments			
`Options`	`CRTM_Options_type`	Scalar or Rank-1 (M)	Options structure containing the option arguments for the CRTM. The dimension should be the same as the input Atmosphere structure.
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Mandatory Output Arguments			
`Atmosphere_K`	`CRTM_Atmosphere_type`	Rank-1 (L) or Rank-2 (L x M)	Structure containing the K-matrix Atmosphere data. NOTE: On ENTRY to this function, the contents of this structure should be defined (e.g. initialized to some value based on the position of this function in the call chain.)
`Surface_K`	`CRTM_Surface_type`	Rank-1 (L) or Rank-2 (L x M)	Structure containing the the K-matrix Surface data. NOTE: On ENTRY to this function, the contents of this structure should be defined (e.g. initialized to some value based on the position of this function in the call chain.)
`RTSolution`	`CRTM_RTSolution_type`	Rank-1 (L) or Rank-2 (L×M)	Solution to the RT problem for each requested channel (L), and each input profile (M). If the input data is scalar, this output is rank-1 (L). If the input data is rank-1 (M), this output is rank-2 (L×M).
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.

Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 29. Description of the `CRTM_K_Matrix()` function arguments and result.

Discussions

(1) The contents of the input structure arguments Atmosphere, Surface, GeometryInfo and ChannelInfo must be appropriately specified. See Section 4 for their definitions and valid assignments.

(2) To request the radiance Jacobinans (derivative of the radiance with respect to input state variables), the user needs to set the following variables as:

RTSolution_K%brightness_temperature = ZERO
RTSolution_K%radiance = ONE

To request the brightness temperature Jacobinans (derivative of the brightness temperature with respect to input state variables), the user needs to set the following variables as:

RTSolution_K%brightness_temperature = ONE
RTSolution_K%radiance = ZERO

(3) The resulting Jacobians are stored in structure Atmospher_K and Surface_K. For instance, Atmospher_K%Temperature(i) is the layer radiance or brightness temperature Jacobian with respect to the temperature at layer *i* and Surface_K%Wind_Speed is the radiance or brightness temperature Jacobian with respect to the surface wind. CRTM also output the Jacobians with respect to the surface emissivity (at observation zenith angle) held in structure RTSolution_K. Table 30 lists the Jacobians returned from the CRTM calculations.

Variable Name	Description
Atmosphere_K%Temperature	
Atmosphere_K%Absorber	
Atmosphere_K%Cloud% Effective_Radius	
Atmosphere_K%Cloud% Water_Content	
Surface_K%Water_Temperature	Ocean surface only
Surface_K%Wind_Speed	Microwave & Ocean surface only
RTSolution_K%Surface_Emissivity	

Table 30. Currently available Jacobians. The units are determined by the units of RTSolution%radiance or RTSolution%brightness_temperature and by the units of the state variables.

5.5.5 CRTM Destruction Routine CRTM_Destroy

The destruction routine CRTM_Destroy is called to deallocate memory occupied by the CRTM coefficient data and the structure variable ChannelInfo. After this call, it is no long valid to call CRTM_forward_Model and CRTM_K_Matrix, until the CRTM initialization routine is called again to load a different (or the same) set of coefficient files and set the ChannelInfo structure.

<u>Calling sequence</u>

```
Error_Status = CRTM_Destroy( ChannelInfo, &
                             Process_Id = Process_Id, &
                             RCS_Id = RCS_Id, &
                             Message_Log = Message_Log )
```

<u>Argument description</u>

The arguments of this function are described in Table 31.

Name	Type	Dimension	Description
Mandatory Input/Output Arguments			
`ChannelInfo`	`CRTM_ChannelInfo_type`	Scalar	ChannelInfo structure. This argument is deallocated in this routine.
Optional Input Arguments			
`Process_Id`	`INTEGER`	Scalar	Set this argument to the MPI process ID that this function call is running under. If MPI is not being used, ignore this argument.
`Message_Log`	`CHARACTER(*)`	Scalar	Filename in which any messages will be logged. Default action is to output messages to screen (stdout).
Optional Output Arguments			
`RCS_Id`	`CHARACTER(*)`	Scalar	String containing the Revision Control System Id field for the module.
Function Result			
`Error_Status`	`INTEGER`	Scalar	Return value indicating the error status. The error codes are defined in the `error_handler` module (see 5.4)

Table 31. Description of the `CRTM_Destroy()` function arguments and result.

Appendix A Structure Member Definition Values

This appendix lists the various public parameters used inside structures in the CRTM. The values listed here are the *only* valid values and will be added to as needed.

Users are *strongly* encouraged to refer to the values below by their symbolic parameter name not a numerical value. The tables below list the parameters in a somewhat logical order, which may or may not be the same as the numerical order. In addition, the actual numerical values assigned to the parameters below can change at any time, whereas the symbolic names will not.

A.1 Atmosphere Structure

Abosrber ID

Absorber (Units)	Parameter Name
H_2O (mass mixing ratio, g/kg)	`H2O_ID`
O_3(volume mixing ratio, ppmv)	`O3_ID`

A.2 Cloud Structure

Valid Cloud type Parameters

Cloud Type	Parameter Name
No cloud	`NO_CLOUD`
Water cloud	`WATER_CLOUD`
Ice cloud	`ICE_CLOUD`
Rain cloud	`RAIN_CLOUD`
Snow cloud	`SNOW_CLOUD`
Graupel cloud	`GRAUPEL_CLOUD`
Hail cloud	`HAIL_CLOUD`

A.3 Surface Structure

A.3.1 Valid Land_Type Parameters

Land Type	Parameter Name	Land Type	Parameter Name
Compacted soil	COMPACTED_SOIL	Grass soil	GRASS_SOIL
Tilled soil	TILLED_SOIL	Broadleaf/Pine forest	BROADLEAF_PINE_FOREST
Sand	SAND	Grass scrub	GRASS_SCRUB
Rock	ROCK	Oil grass	OIL_GRASS
Irrigated low vegetation	IRRIGATED_LOW_VEGETATION	Urban concrete	URBAN_CONCRETE
Meadow grass	MEADOW_GRASS	Pine brush	PINE_BRUSH
Scrub	SCRUB	Broadleaf brush	BROADLEAF_BRUSH
Broadleaf forest	BROADLEAF_FOREST	Wet soil	WET_SOIL
Pine forest	PINE_FOREST	Scrub soil	SCRUB_SOIL
Tundra	TUNDRA	Broadleaf(70)/Pine(30)	BROADLEAF70_PINE30

A.3.2 Valid Snow_Type Parameters

Snow Type	Parameter Name
New snow	New_Snow
Old snow	Old_Snow

A.3.3 Valid WMO Sensor ID Parameters Used in the Surface Emissivity Model Calculations

Surface type	Sensor Name	Sensor Channels	WMO Sesnor ID
Ice, Snow	AMSUA	1, 2, 3, 4	WMO_AMSUA
Ice, Snow	AMSUB	1, 2	WMO_AMSUB
Ice, Snow	AMSRE	1 to 12	WMO_AMSRE
Ice, Snow	SSMI	1 to 7	WMO_SSMI

Appendix B Sensor List

The sensors and channels currently covered by the CRTM are listed below.

Microwave Sensors

Sensor name	Satellite name	Sensor descriptor	# of channels	Channel number
AMSR-E	AQUA	amsre_aqua	12	1,2,3,4,5,6,7,8,9,10,11,12
AMSU-A	AQUA	amsua_aqua	15	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
AMSU-A	NOAA-15	amsua_n15	15	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
AMSU-A	NOAA-16	amsua_n16	15	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
AMSU-A	NOAA-17	amsua_n17	15	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
AMSU-A	NOAA-18	amsua_n18	15	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
AMSU-B	NOAA-15	amsub_n15	5	1,2,3,4,5
AMSU-B	NOAA-16	amsub_n16	5	1,2,3,4,5
AMSU-B	NOAA-17	amsub_n17	5	1,2,3,4,5
ATMS	NPOESS-C1	atms_c1	22	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
HSB	AQUA	hsb_aqua	4	1,2,3,4
MHS	NOAA-18	mhs_n18	5	1,2,3,4,5
MSU	TIROS-N	msu_n05	4	1,2,3,4
MSU	NOAA-06	msu_n06	4	1,2,3,4
MSU	NOAA-07	msu_n07	4	1,2,3,4
MSU	NOAA-08	msu_n08	4	1,2,3,4
MSU	NOAA-09	msu_n09	4	1,2,3,4
MSU	NOAA-10	msu_n10	4	1,2,3,4
MSU	NOAA-11	msu_n11	4	1,2,3,4
MSU	NOAA-12	msu_n12	4	1,2,3,4
MSU	NOAA-14	msu_n14	4	1,2,3,4
SSM/I	DMSP-13	ssmi_f13	7	1,2,3,4,5,6,7
SSM/I	DMSP-14	ssmi_f14	7	1,2,3,4,5,6,7
SSM/I	DMSP-15	ssmi_f15	7	1,2,3,4,5,6,7
SSMIS	DMSP-16	ssmis_f16	24	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
SSM/T-1	DMSP-13	ssmt1_f13	7	1,2,3,4,5,6,7
SSM/T-1	DMSP-15	ssmt1_f15	7	1,2,3,4,5,6,7
SSM/T-2	DMSP-14	ssmt2_f14	5	1,2,3,4,5
SSM/T-2	DMSP-15	ssmt2_f15	5	1,2,3,4,5
WindSat	Coriolis	windsat_coriolis	16	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

Infrared sensors

Sensor name	Satellite name	Sensor descriptor	# of channels	Channel number
AIRS	AQUA	airs_aqua	2378	1 to 2378
AIRS	AQUA	airsSUBSET_aqua	281	1,6,7,10,11,15,16,17,20,21,22,24,27,28,30,36,39,40,42,51,52,54,55,56,59,62,63,68,69,71,72,73,74,75,76,77,78,79,80,82,83,84,86,92,93,98,99,101,104,105,108,110,111,113,116,117,123,124,128,129,138,139,144,145,150,151,156,157,159,162,165,168,169,170,172,173,174,175,177,179,180,182,185,186,190,192,198,201,204,207,210,215,216,221,226,227,232,252,253,256,257,261,262,267,272,295,299,300,305,310,321,325,333,338,355,362,375,453,475,484,497,528,587,672,787,791,843,870,914,950,1003,1012,1019,1024,1030,1038,1048,1069,1079,1082,1083,1088,1090,1092,1095,1104,1111,1115,1116,1119,1120,1123,1130,1138,1142,1178,1199,1206,1221,1237,1252,1260,1263,1266,1285,1301,1304,1329,1371,1382,1415,1424,1449,1

				455,1466,1477,1500,1519,1538,1545,1565,1574,1583,1593,1614,1627,1636,1644,1652,1669,1674,1681,1694,1708,1717,1723,1740,1748,1751,1756,1763,1766,1771,1777,1780,1783,1794,1800,1803,1806,1812,1826,1843,1852,1865,1866,1868,1869,1872,1873,1876,1881,1882,1883,1911,1917,1918,1924,1928,1937,1941,2099,2100,2101,2103,2104,2106,2107,2108,2109,2110,2111,2112,2113,2114,2115,2116,2117,2118,2119,2120,2121,2122,2123,2128,2134,2141,2145,2149,2153,2164,2189,2197,2209,2226,2234,2280,2318,2321,2325,2328,2333,2339,2348,2353,2355,2357,2363,2370,2371,2377
AVHRR/2	TIROS-N	avhrr2_n05	2	3,4
AVHRR/2	NOAA-06	avhrr2_n06	2	3,4
AVHRR/2	NOAA-07	avhrr2_n07	3	3,4,5
AVHRR/2	NOAA-08	avhrr2_n08	2	3,4
AVHRR/2	NOAA-09	avhrr2_n09	3	3,4,5
AVHRR/2	NOAA-10	avhrr2_n10	2	3,4
AVHRR/2	NOAA-11	avhrr2_n11	3	3,4,5
AVHRR/2	NOAA-12	avhrr2_n12	3	3,4,5
AVHRR/2	NOAA-14	avhrr2_n14	3	3,4,5
AVHRR/3	NOAA-15	avhrr3_n15	3	3,4,5
AVHRR/3	NOAA-16	avhrr3_n16	3	3,4,5
AVHRR/3	NOAA-17	avhrr3_n17	3	3,4,5
AVHRR/3	NOAA-18	avhrr3_n18	3	3,4,5
HIRS/2	TIROS-N	hirs2_n05	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-06	hirs2_n06	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-07	hirs2_n07	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-08	hirs2_n08	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-09	hirs2_n09	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-10	hirs2_n10	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-11	hirs2_n11	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-12	hirs2_n12	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/2	NOAA-14	hirs2_n14	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/3	NOAA-15	hirs3_n15	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/3	NOAA-16	hirs3_n16	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/3	NOAA-17	hirs3_n17	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
HIRS/3	NOAA-18	hirs3_n18	19	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
IMAGER	GOES-08	imgr_g08	4	2,3,4,5
IMAGER	GOES-09	imgr_g09	4	2,3,4,5
IMAGER	GOES-10	imgr_g10	4	2,3,4,5
IMAGER	GOES-11	imgr_g11	4	2,3,4,5
IMAGER	GOES-12	imgr_g12	4	2,3,4,6
MODIS	AQUA	modis_aqua	16	20,21,22,23,24,25,27,28,29,30,31,32,33,34,35,36
MODIS	TERRA	modis_terra	16	20,21,22,23,24,25,27,28,29,30,31,32,33,34,35,36
SOUNDER	GOES-08	sndr_g08	18	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
SOUNDER	GOES-09	sndr_g09	18	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
SOUNDER	GOES-10	sndr_g10	18	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
SOUNDER	GOES-11	sndr_g11	18	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
SOUNDER	GOES-12	sndr_g12	18	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18